



TITLE:

2分検索木上のセレクティブトラバースに関するある改良 (情報科学の数学的基礎理論と応用)

AUTHOR(S):

菊野, 亨; 楠本, 博巳

CITATION:

菊野, 亨 ...[et al]. 2分検索木上のセレクティブトラバースに関するある改良 (情報科学の数学的基礎理論と応用). 数理解析研究所講究録 1979, 353: 87-96

ISSUE DATE:

1979-04

URL:

<http://hdl.handle.net/2433/104424>

RIGHT:

2分検索木上のセレクティブトラバース に関するある改良

広島大学 工学部 菊野 亨
楠本 博巳

1. まえがき

2分検索木は格納されているレコードへのアクセス及びファイルの変更が容易であるので、大規模なファイルを構成する際の重要な技法としてよく知られている⁽¹⁾。通常、2分検索木の各節点 v_i には (k_i, P_i) の2項組のデータが保持される。ここで k_i はキーと呼ばれ、 P_i は対応するレコードが実際に格納されているメモリ上の領域へのポインタとする。レコードへのアクセスに関する興味ある問題の1つに、区域条件 $C_j = (l_j, r_j)$ (l_j, r_j は共に非負整数で、 $l_j < r_j$)の系列 $L = C_1, C_2, \dots, C_n$ が与えられ、少なくとも1つの C_j に対し $l_j < k_i < r_j$ なるキーをもつ2分検索木上の全ての節点を、その途中の過程で参照する節点の数を最小にしてアクセスするというセレクティブトラバースの問題がある。

Driscollらは2分検索木上の各節点に3ビットのマーカ

フィールドを設け、7つのマークを用いることにより、上述の問題を能率よく解いている⁽¹⁾。本論文では、若干の(時間的な)オーバーヘッドを犠牲にすれば、4つのマークで十分であること、すなわちマークフィールドを2ビットにまで減じることが可能であることを示す。

2. 準備

[定義1] 2分検索木 T の各節点 v_i に格納されているデータはポインタフィールド、データフィールド及びマークフィールドの3つに分かれている。

(i) ポインタフィールドには、左部分木へのポインタ LEFT, 右部分木へのポインタ RIGHT が含まれている。

(ii) データフィールドには2項組のデータ (k_i, P_i) が保持されている。ここで k_i はキーと呼ばれ N (N は非負整数全体の集合)の要素である。 P_i は対応するレコードが格納されている領域へのポインタであるが、今の場合、 P_i の値については関心がないので省くことにする。

(iii) マークフィールドは3ビットの容量をもつ。

[定義2] 区域条件 C_j とは2項組 (l_j, r_j) のことである。但し、 l_j と r_j は共に N のある要素であって、関係式 $l_j < r_j$ が成立するものとする。区域条件の系列 C_1, C_2, \dots, C_n を L と表す。 L において $l_p \leq l_q < r_p$ あるいは $l_p < r_q \leq r_p$ を満たす相

異なる添字 P , q が存在するなら, L にはオーバーラップが存在するという.

〔定義 3〕 2分検索木 T 上の節点 v_i のキー k_i と区域条件 $C_j = (l_j, r_j)$ に対し, $l_j < k_i < r_j$ が成立するなら, 節点 v_i は区域条件 C_j を満たすという. T と区域条件の系列 L に対し, L に属する区域条件の少なくとも1つを満たす節点の集合を $T(L)$ と表す.

〔定義 4〕 2分検索木 T と区域条件 L が与えられ, その途中の過程で参照する節点の数を最小に押えて $T(L)$ に属する節点を全てアクセスする問題をセレクトィブトラバース (selective traverse) の問題と呼ぶ. 以降では単に問題 ST と記すことにする.

Driscoll らは, 区域条件の系列 L にオーバーラップが存在しない場合について問題 ST を解くアルゴリズム $SELECT$ (以降ではアルゴリズム S と呼ぶ) を与えている. アルゴリズム S では, 2分検索木 T の各節点にあるマークフィールド (3ビット) 上で識別可能な7つのマーク i, l, r, a, s, n, f を用いており, スタック, キューなどはいない.

3. 改良

アルゴリズム S に対し, 使用するマークの数を7から4に

減じる，すなわちマーカフィールドのビット数を1ビット減じる事が可能であることを示す。

3. 1 基本方針

ここで提案するアルゴリズムはDriscollらと基本的に同じ立場をとり，スタックは用いず，各節点ごとに付加したマーカフィールドの値を参照しながら2分探索木了をたどる。但しマーカフィールドを1ビット減じて2ビットとするため，使用するマーカも i ， l ， r ， a の4つとする。改良アルゴリズムは次の2つのフェーズに分かれている。

フェーズⅠ： \mathcal{L} と \mathcal{L} に対し，集合 $\mathcal{L}(\mathcal{L})$ に属する節点を含む部分木の周囲に位置する節点に対し，マーカ l ， r と a を境界記号として書き込む。

フェーズⅡ：フェーズⅠで準備した各節点のマーカと，必要ならば左子節点および右子節点のマーカとを調べながら上をたどっていった， $\mathcal{L}(\mathcal{L})$ に属する節点だけに必要な処理を行う。

フェーズⅠはマーカ s を l で代用することを除けば，アルゴリズム S とほぼ同じ。フェーズⅡでは，アルゴリズム S で用いていたマーカ s を l あるいは r で，マーカ n を i でそれぞれ表しておいて，ある節点 v において（アルゴリズム S で行ったマーカの区別が）必要となれば v の左子節点および右

子節点のマークを調べることにより，アルゴリズム S のステップⅡの動きを正しくシミュレートしている。

3. 2 改良アルゴリズムの概要（マークの説明）

マークの説明を行うことによって改良アルゴリズム，特にフェーズⅡの概要を述べる。先ず，アルゴリズム S に存在したが，改良アルゴリズムでは省いたマークについて簡単に説明する。次に 2，3 の記号を準備しておく。

$lson(v)$ ：節点 v の左子節点 (left son)。

$lsub(v)$ ：節点 v の左子節点を頂点とする部分木。

$rson(v)$ ：節点 v の右子節点 (right son)。

$rsub(v)$ ：節点 v の右子節点を頂点とする部分木。

各マークの設定されている節点を v とする。

マーク S：改良アルゴリズムにおいては，各節点 v に対し先ず $lsub(v)$ 内をたどり，その後で $rsub(v)$ 内をたどることとし，マーク S をマーク l で代用する。

マーク n ：マーク i あるいは a の n への変換を省き，直接，処理を行う。マーク n をマーク i で代用する。

マーク f ：節点 v の親に移る時には， $lson(v)$ ， $rson(v)$ のマークが共に i となっていることに注目して，マーク r あるいは l で代用する。なお $lson(v)$ と $rson(v)$ のマークを調べることにより， f の復元は可能（詳細は後述のマ

ーカ r を参照)。

次に、改良アルゴリズムで使用する4つのマークについて詳しく述べる。アルゴリズム S のフェーズⅡの実行前における、マーク r の節点 v の周りの状況は図1の通りである。又マーク l の節点 v 、マーク a の節点 v の周りの状況はそれぞれ図2、図3の通りである。なお図中のマークは $\alpha \in \{r, a, s\}$, $\beta \in \{l, a, s\}$ とする。

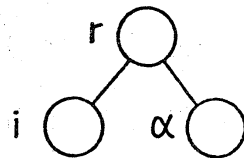


図1 マーク r の節点

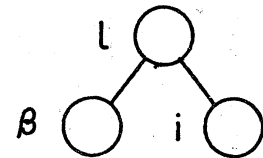
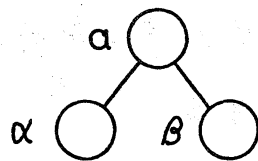
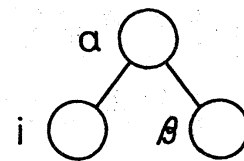


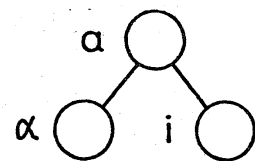
図2 マーク l の節点



(1) 場合1



(2) 場合2



(3) 場合3

図3 マーク a の節点

- ①マーク i ：基本的にはアルゴリズム S と同じ。
 - ②マーク r ：アルゴリズム S のフェーズⅡでは先ず節点 v のマーク r (図1参照)を f に変換し、 $rson(v)$ に移り、 $rsub(v)$ 内での実行が終了すると、 $rson(v)$ のマーク α を i に変換し図4の形で(マーク f の) v に再び戻ってくる。
- そこで改良アルゴリズムのフェーズⅡでは、 v のマーク r を(f に)変換することなく、常に $rson(v)$ に移る。必要

なら再び v に戻ってきた時点で $rson(v)$ のマークが i かどうかを調べることによって、 r と f の区別は可能である。

③マーク l ：マーク s の説明で述べた様に、改良アルゴリズムではマーク s を l で代用するので、図5の形と図2の形のマーク l の節点 v を同時に取り扱うことが必要となる。図でマークは $\alpha' \in \{r, a, l\}$, $\beta' \in \{l, a\}$ である。

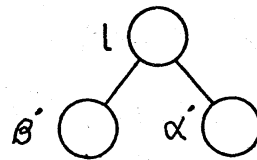
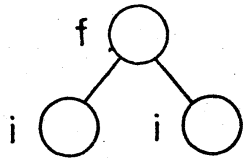


図4 マーク f の節点

図5 マーク l の説明図

改良アルゴリズムのフェーズⅡでは、節点 v のマーク l を r に一旦変換しておいて、 $lson(v)$ に移る。こうして、 $lsub(v)$ 内での実行が終了して再び v に戻った時、(②で述べた手順に従えば)図2なら図4(f は r となっている)の形に、図5なら図1の形にそれぞれ遷移して、 l と s の区別が正しく実現される。

④マーク a ：アルゴリズム S のフェーズⅡではマーク a は図3の形で現れる。この時、マーク i の節点($lson(v)$ あるいは $rson(v)$)を頂点とする部分木内には、ある区域条件を満たす(マーク i の)節点だけが存在する。改良アルゴリズムのフェーズⅡでは、 $lson(v)$ のマークが i か否か判定し、 i なら $lsub(v)$ に対しLEFTINNERを呼ぶ。

LEFTINNERの実行の最後に($v \in \mathcal{L}(L)$ なので)
 v に対し処理を行い, v のマーク a を r に変換して v に戻り,
 以降は②のマークの手順に帰着される. i でないなら v の
 マーク a を i に変換し $lson(v)$ に移る. $lsub(v)$ 内での
 実行が終了して v に戻ると, v に対し処理を行う. 次に $rson$
 (v) のマークが i か否かを判定し, i なら v のマーク i を
 a に変換し, $rsub(v)$ に対しRIGHTINNERを呼び,
 RIGHTINNERの実行の最後に v のマーク a を r に変
 換する. 以降は②のマークの手順に帰着される.

4. 評価

評価に必要な記号の説明をしておく.

C: 定数(例えば区域条件に含まれる l_i と r_i)と2分検
 索木内の節点のキーの大小比較演算.

T: 2分検索木内の節点のマークの判定演算.

P: 2分検索木内の節点(に含まれるデータ)に対し実際
 に処理を行う演算で, アルゴリズムではPROCESS
 Sという命令で記述している演算.

次に N 個の節点を含む完全2分検索木 T とオーバーラップの
 ない区域条件のリスト $L = C_1, C_2, \dots, C_n$ に対する問題STにつ
 いて, アルゴリズムの実行時間に関する評価を行う. 以下で
 はいずれも最悪の場合を取扱うものとする. 但し, $\#(C_k)$

は区域条件 C_k を満たす節点の数とする.

Driscoll らのアルゴリズム S の場合⁽¹⁾

$$\begin{aligned} & [6n \log N + 4 \sum_{k=1}^n \#(C_k)] T \\ & + [2n \log N] C + [\sum_{k=1}^n \#(C_k)] P \cdots (1) \end{aligned}$$

改良アルゴリズムの場合

$$\begin{aligned} & [8n \log N + 4 \sum_{k=1}^n \#(C_k)] T \\ & + [2n \log N] C + [\sum_{k=1}^n \#(C_k)] P \cdots (2) \end{aligned}$$

(1) 式と (2) 式を比較すれば改良アルゴリズムの場合, アルゴリズム S に比べて, $(2n \log N)$ だけ余計に時間がかかることが分かる. しかし, 上記の3つの演算 C, T, P の内, 演算 T に要する時間は他の2つに比べ比較的少ないと予想される⁽¹⁾. したがって, この時間的損失はマーカフィールドの1ビットの削減により, 十分償われると思われる.

5. むすび

Driscoll らが与えていたアルゴリズム S に対し, 1ビットの削減が可能であることを示した. なお, 改良アルゴリズムの場合の評価を厳密に行えば, $0 \leq P \leq 1$ なるある数 P に対し, $[2n \log N - P \sum_{k=1}^n \#(C_k)] T$ だけのオーバーヘッドとなり, 改良アルゴリズムの方がアルゴリズム S よりも速くなることも多くの場合について実測されている. フェーズ II をフェーズ I に組み込んだアルゴリズムも得ており, 別の機

会に報告する予定である.

文 献

- (1) J.R.Driscoll and Y.E.Lien: "A selective traversal algorithm for binary search trees", CACM, 21, 6, p.445 (June 1978).